EVOLUTION OF ADA TECHNOLOGY IN THE FLIGHT DYNAMICS AREA: DESIGN PHASE ANALYSIS

DECEMBER 1988

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-88-003

EVOLUTION OF ADA TECHNOLOGY IN THE FLIGHT DYNAMICS AREA: DESIGN PHASE ANALYSIS

DECEMBER 1988

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC, Systems Development Branch
The University of Maryland, Computer Sciences Department
Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The authors of this document are

Kelvin L. Quimby (Computer Sciences Corporation)
Linda Esker (Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

The software engineering issues related to the use of the Ada programming language during the design phase of an Ada project are analyzed. Discussion shows how an evolving understanding of these issues is reflected in the design processes of three "generations" of Ada projects.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

EXECUTIVE SUMMARY

During the past 4 years, the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) with support from Computer Sciences Corporation (CSC) has been using the Ada programming language in five different projects. The first two of these projects, the Flight Dynamics Analysis System (FDAS) and the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY), were research-oriented projects. The three Ada projects that followed are production satellite simulation systems for use in support of actual missions. The Geostationary Operational Environmental Satellite-I (GOES-I) will be supported by the GOES-I Dynamics Simulator (GOADA) and the GOES-I Telemetry Simulator (GOESIM). The Upper Atmosphere Research Satellite (UARS) will be supported by the UARS Telemetry Simulator (UARSTELS).

This report analyzes the software engineering issues related to the use of the Ada programming language during the design phase of an Ada project and discusses how an evolving understanding of these issues is reflected in the design processes of three "generations" of Ada projects: FDAS and GRODY, GOADA and GOESIM, and UARSTELS. The following points summarize this analysis:

- The GRODY project introduced object-oriented design and the entity-diagram notation, both of which have been adopted by all of the other Ada projects. However, GRODY was inadvertently designed in a way that required expending considerable effort in restructuring very large software modules into smaller ones so that they could be reused in subsequent Ada projects. The mathematical algorithms from GRODY continue to be widely used on all subsequent simulator projects.

- The FDAS project initiated the extensive use of small, separately compiled software components as a mechanism to modularize loosely coupled design elements and the use of type packages and subsystems as design entities.

- The GOADA project formalized and expanded the use of subsystems as a design entity, examined the best features of the GRODY and FDAS entity diagrams, and incorporated these into an improved form of design notation; it was the first project to develop a compiled design.

- The UARSTELS project, which greatly increased the use of Ada generic packages, was the first project to emphasize during design the development of software components that could be reused on subsequent projects without changes.

- The incremental design approach used on the production Ada projects meshes well with the existing preliminary design review (PDR)/critical design review (CDR) approach that has been utilized on traditional FORTRAN systems. The modifications made to this approach that are Ada-specific include the use of the entity diagram notation for both the preliminary design report (for PDR) and the detailed design document (for CDR). In addition, for the PDR most of the Ada package specifications to be used in the system are compiled, and for the CDR the matching Ada package implementations with program design language (PDL) are developed and compiled to the point that the code associated with the design can be linked into an executable image.

- Additional thought needs to be given to how to more effectively exploit features of Ada to maximize the amount of reusability of Ada software design components from one simulation project to another. Consideration should be given as to how to develop these design entities so that they are

reusable on larger systems, such as Attitude Ground Support Systems (AGSSs), and even on very large-scale systems, including the Space Station project.

SECTION 1 - INTRODUCTION AND BACKGROUND

1.1 INTRODUCTION

This report is the first of a series of four reports describing the growth of Ada technology at Goddard Space Flight Center (GSFC) and Computer Sciences Corporation (CSC). This technology continues to evolve through an accumulating experience base gained from the development efforts associated with several past and current Ada projects at GSFC/CSC. This first report is primarily concerned with the evolving understanding of software engineering issues related to the use of Ada during the design phase of a project. Additional reports are scheduled to follow, one covering the evolution of Ada technology in implementation and one covering testing. A final summary report on Ada project characteristics will conclude the series.

The first section of the report (Section 1) provides background information, including a brief description of each of the Ada projects studied. Section 2 discusses the software engineering issues related to designing a system in Ada and how a growing understanding of these issues has been incorporated in the design documentation generated for each new project. Section 3 presents the general project characteristics of each of the Ada simulator systems, including Ada experience and training of project personnel, Ada software metrics, and effort and productivity measures. Section 4 summarizes the lessons learned in the design phases of all of the Ada projects and presents a number of recommendations to designing future systems in Ada.

1.2 BACKGROUND

Since 1960, GSFC has relied heavily on FORTRAN in developing software systems for mission analysis, satellite simulations,

and Attitude Ground Support Systems (AGSSs). In 1985, GSFC and CSC began using the Ada programming language on two different types of projects. The first project to use Ada was an in-house, research-oriented project called the Flight Dynamics Analysis System (FDAS). The FDAS project developed a software reconfiguration tool for use by National Aeronautics and Space Administration (NASA) analysts to experiment with different algorithms for solving spacecraft orbit and attitude analytical problems. Shortly after the decision was made to use Ada as the implementation language for FDAS, a second Ada project, called the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY), was started. This experimental Ada development project for use as part of the ground support system for the GRO satellite was the first attempt at using Ada for the type of mathematically oriented system typically developed in this environment. A corresponding version of this simulator, the GRO Simulation Systems (GROSS), was developed in FORTRAN. The two simulator projects, GRODY and GROSS, were developed in two different languages to gain some understanding of the impact Ada is likely to have on the development of software in the flight dynamics area (Brophy et al., 1987; Godfrey and Brophy, 1987, 1989; Seigle and Shi, 1988).

Both FDAS and GRODY can be characterized as research and development efforts because they were nonoperational projects for which a considerable amount of experimentation and/or prototyping was utilized in their development. Since both of these projects were first-time Ada projects for this environment, they are considered here as "first generation" Ada technology.

With the development experience gained from these first-time Ada projects, the decision was made to use Ada as the implementation language on two production satellite simulation

projects to be used in support of Geostationary Operational Environmental Satellite-I (GOES-I). The first of these two projects is the GOES-I Attitude Dynamics Simulator (GOADA), which began on May 30, 1987, and completed detailed design on March 19, 1988. The second project is the GOES-I Telemetry Simulator (GOESIM), which began on September 5, 1987, and completed detailed design on April 30, 1988. Unlike GRODY, these projects are required to adhere very closely to project schedules. They can be viewed as "second generation" Ada projects because they have drawn heavily on the lessons learned from both FDAS and GRODY in their design.

The most recent Ada project is also a production system to be used in support of the Upper Atmosphere Research Satellite (UARS). The UARS Telemetry Simulator (UARSTELS) project began on February 13, 1988, and detailed design was completed on September 10, 1988. Because this project has emphasized improving the designs of GOADA and GOESIM, it can be viewed as a "third generation" Ada project.

Of the four satellite simulation projects, two are dynamics simulators, and two are telemetry simulators. For the purposes of comparison, the objective information presented in Section 3 of this document will be drawn from these four projects. For the analysis of this objective data, it is necessary to first discuss how an understanding of the technical issues associated with designing software systems in Ada has evolved over the history of all five Ada projects. This type of subjective analysis is based on discussions and interviews with developers from the five projects, from questionnaires filled out by these developers, and from published literature on both Ada and software engineering practices, in general.

SECTION 2 - ADA DESIGN METHODOLOGY

2.1 OBJECT-ORIENTED DESIGN AND ENTITY DIAGRAMS

The GRODY team intensely investigated various design methodologies that could be used in developing a medium-sized Ada software system, and eventually adopted a modified version of object-oriented design (Agresti et al., 1986; Godfrey and Brophy, 1987; Seidewitz and Stark, 1986). This decision continues to exert a strong and growing influence over subsequent Ada projects and, as such, represents one of the more important contributions GRODY has made to the evolution of Ada technology at GSFC/CSC.

Closely related to the design methodology is the issue of the representation of the design on paper. The previous design study has noted that documenting an Ada design, in particular when object-oriented design techniques are used, requires different design products than typically used for FORTRAN systems (Godfrey and Brophy, 1987). The object or entity diagram notation introduced by GRODY for graphically representing Ada designs has been adopted by the subsequent Ada projects, each of which has introduced further refinements and enhancements to the graphic notation. These design diagrams will be used throughout this section to illustrate the various points being made.

The GRODY team developed a notation for representing basic Ada components based on ideas from George Cherry's process abstraction methodology (PAMELA) (Cherry, 1985) and Grady Booch's object-oriented design (Booch, 1983). Bubble/rectangles (bub-tangles) are used to represent packages; rectangles are used to represent subprograms; and parallel lines are used to represent state data (Figure 2-1). These symbols continue to be used on the current projects.

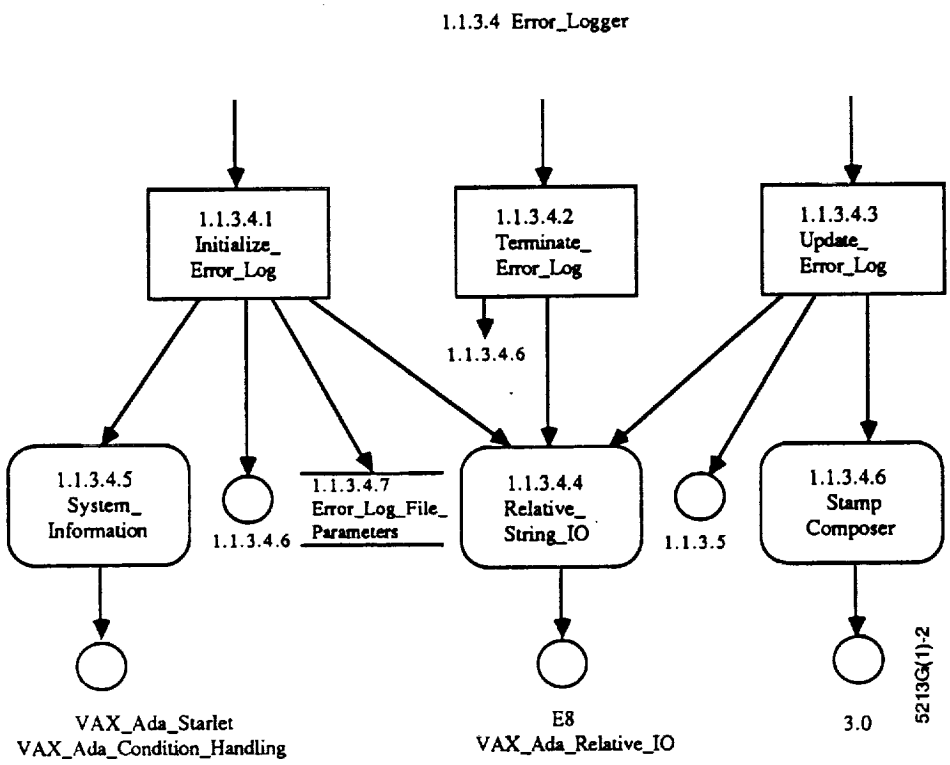

Figure 2-1. Example of Design Diagram From GRODY

The major advantage of the graphic representation introduced by GRODY is the ease of representing the design of an entire, medium-sized project within a three-ring notebook, with a simple notation that is easy to understand and draw by hand. The hierarchical structure of a system is indicated by the use of leveled diagrams, with off-page connectors illustrating the top-down relationships among components that span page boundaries.

The FDAS team felt that additional information was needed on these diagrams to specify the bottom-up relationships among components that spanned page boundaries. For example, Figure 2-1 from the GRODY system description document illustrates that it cannot be determined from looking at the design diagram what particular entities reference the operations Initialize_Error_Log, Terminate_Error_Log, and Update Error_Log. This type of information is important because the design notebook is, in reality, a medium for communication between developers. Thus, when the individual in charge of developing Error_Logger adds, deletes, or modifies any of the formal parameters associated with the three visible operations mentioned above, that individual knows what components outside of his or her domain are affected and can notify the individual(s) responsible for developing those components. FDAS added this type of information to their design notation by providing the page connector symbol and a number that indicated the location of the component(s) that reference the entity on the diagram (Figure 2-2).

The entity diagram notation introduced by GRODY has provided a firm foundation for refinements and enhancements that have been incrementally introduced by subsequent Ada projects and can be expected to further evolve as more advanced features of Ada are adopted on subsequent projects.

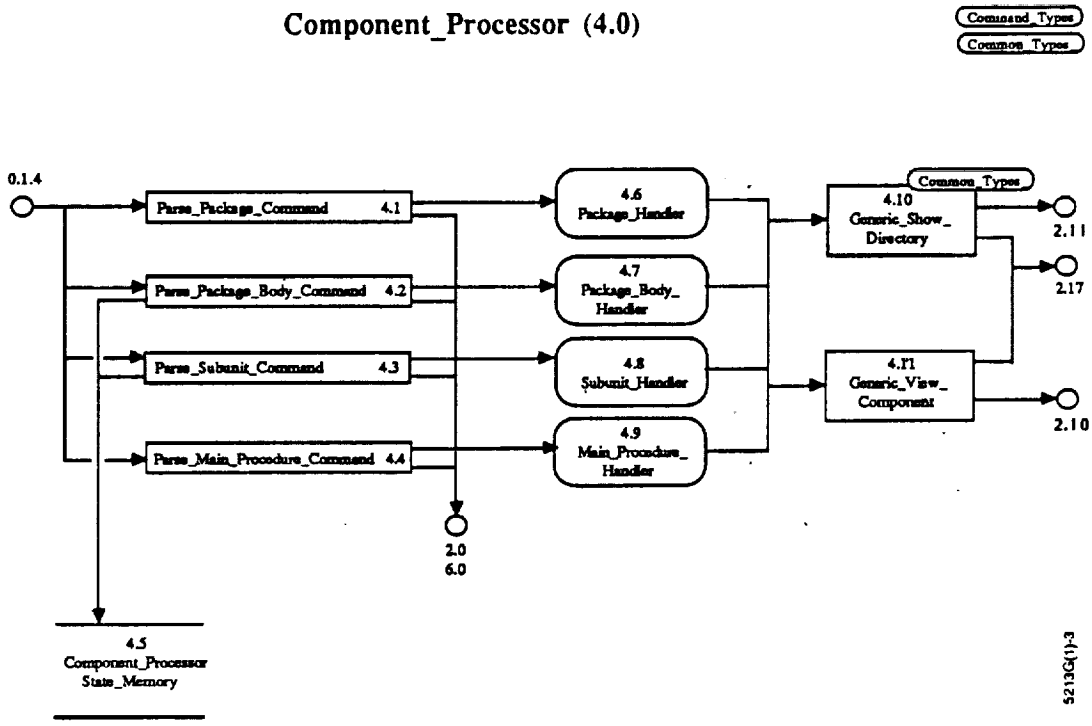

Figure 2-2. Example of Off-Page Connectors for Design Diagrams Used in FDAS

2.2 SUBSYSTEMS

The idea of Ada subsystems, a very important design concept, was introduced after GRODY. A subsystem in Ada is an abstract entity that is composed of a number of Ada packages, subprogram compilation units, and possibly other lower-level subsystems (Booch, 1987). It is an abstract entity because no specific Ada component is used to represent a subsystem. A subsystem is analogous to a package because generally some of the constituent components that make up the subsystem provide visible operations to users of the subsystem, whereas other constituent components are hidden (Figure 2-3).

Subsystems were first used as a design entity in the FDAS research project, and the concept was formalized and expanded by the GOADA project. Subsystems were not used in GRODY because of the team's interpretation of what an object is in object-oriented design. To GRODY, the Ada package was the de-facto implementation vehicle for objects. Thus, all objects were represented as a single package. If the object were too large to be implemented in its entirety by a single package, it could be decomposed into lower-level packages, but the operations on the object were constrained to be implemented within a single, top-level package. Thus, in GRODY, an object always had a single package as its root.

Using this approach, the decomposition of an object into lower-level packages required that the specifications of these packages had to be tied to the root package through one of the following:

1. Physically nested within the root package specification

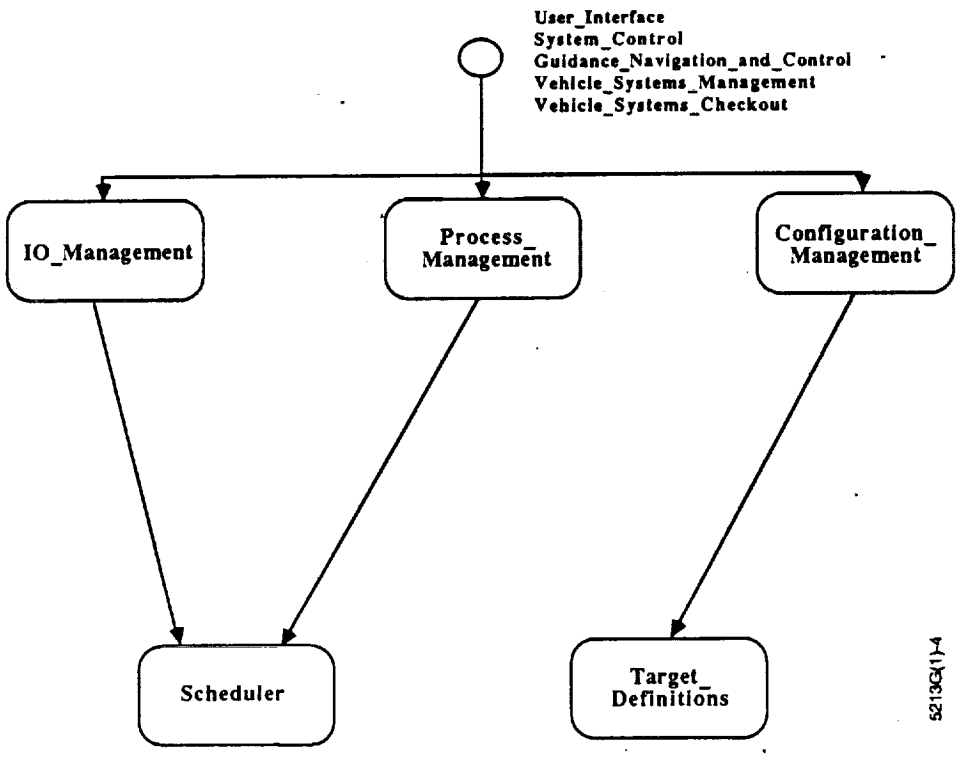

Figure 2-3. Example of Subsystem Concept (Booch, 1987)

2. Implemented as library units or physically nested within the root package specification or body but accessed by components outside of the root package via call-throughs

3. Implemented as package specifications that contain Ada renames statements for indicating visible operations

Utilities, the entity in GRODY that illustrates approach (1), is a package instantiation of Generic_Utilities composed of a number of lower-level packages, including Math_Functions, Linear, and Attitude_Math. These three math packages were physically nested within the specification of Generic_ Utilities, and their operations could only be accessed through the instance name Utilities, such as Utilities.Math_ Functions.Sin, Utilities.Linear.Unit_Vector, etc. With this approach, all operations or services provided by the sum of the constituent packages of the object have to be provided within the single root package specification of the object, even if those operations or services are organized into lower-level packages. Thus, for Generic_Utilities, the specifications for a total of 49 procedures and functions were provided within the single package specification Generic_Utilities.

A number of problems exist with this approach that were not apparent to the GRODY team at the time the system was designed (Clarke et al., 1980). The major problem addressed in this section is that this approach does not scale-up well from small programs (Booch, 1987). A utilities object designed this way in a large-scale Ada system, such as the Space Station, might contain several dozen math and other utility-package specifications nested within the specification for the object Utilities and, therefore, hundreds of

subprograms specified physically within this one large package. Since almost every component in such a system would have to access one or more packages within Utilities, hundreds of thousands of lines of Ada code would have to be recompiled every time the slightest modification were made to the specification of Utilities, even though that modification may have been made for the benefit of only a few packages within the entire system (such as changing the mode of a single parameter within a single routine within one of the nested package specifications). In structured design terminology, such a system contains modules that are tightly coupled (Myers, 1978).

In Build 1 of FDAS, the package Utility was designed just as on GRODY in that its constituent packages were physically nested within the package specification of Utility. However, by Build 2 of FDAS, Utility had evolved into the kind of abstract entity described above (Figure 2-4). The four packages nested within the package Utility were extracted, renamed, and compiled as library units (Figure 2-5). The package Utility itself was discarded, but the entity was retained as a design concept. In other words, Utility became a subsystem as defined by Booch (1987).

The problem with the design diagram from FDAS is that it is not readily apparent from the diagram which entities are packages and which are subsystems. The trailing underscore in the name Utility_ was meant to indicate that it represented a collection of packages, with the convention that each package in this collection would begin with this name (i.e., Utility_Host_Command_Handler, Utility_Log_File_Handler, etc.). However, this was found to be unnecessarily restrictive because this forces verbose package names. (The package Configuration_Management in the subsystem Flight

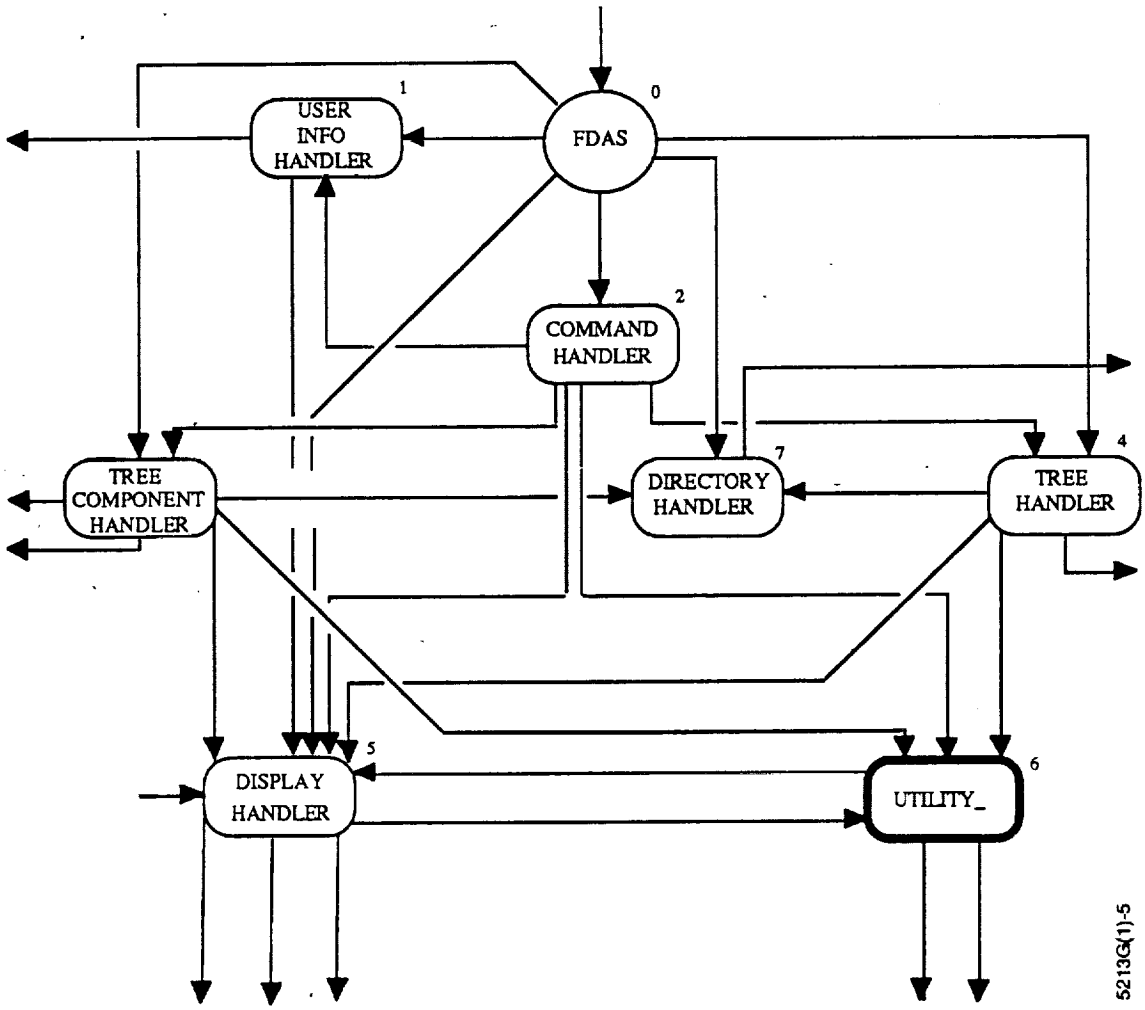

Figure 2-4. Example of Subsystem (Utility_) From Build 2 of FDAS

UTILITY_

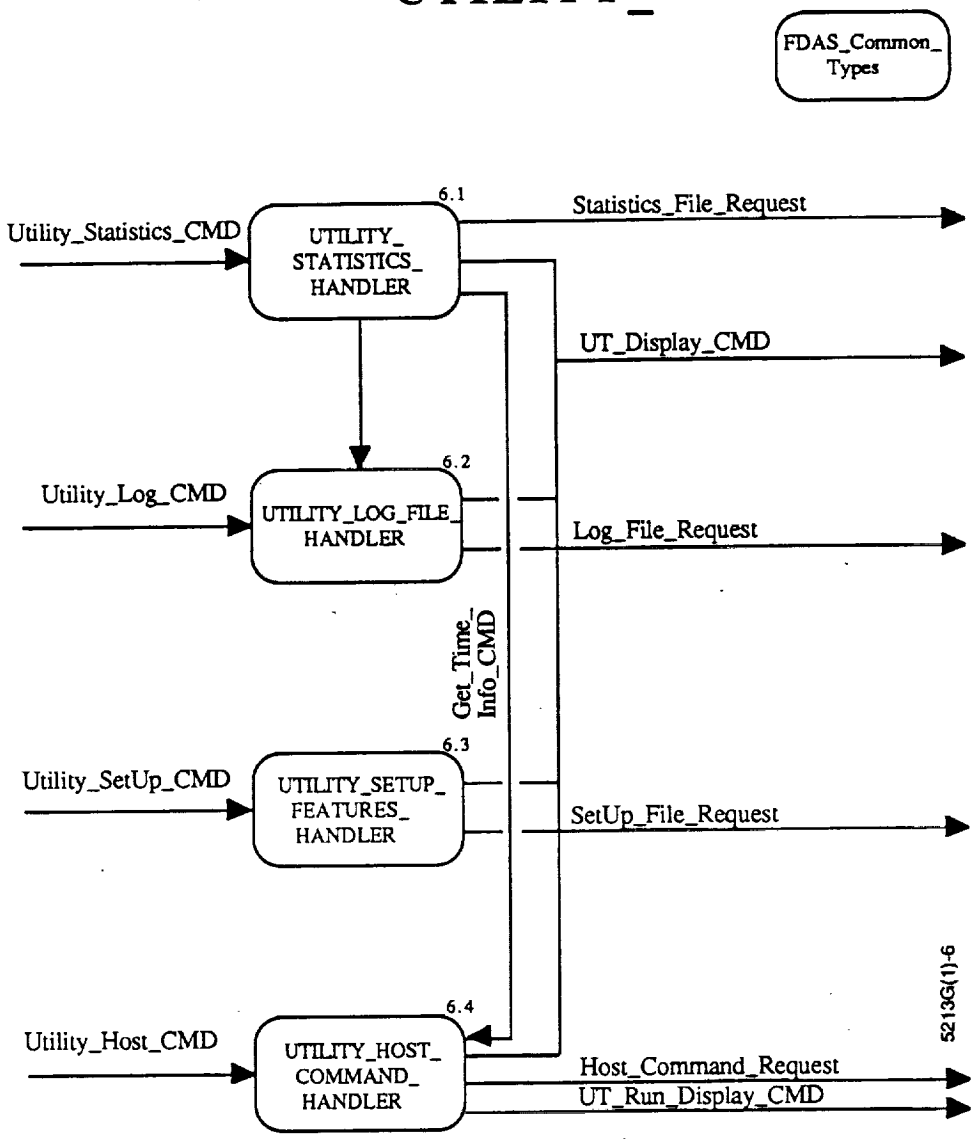

Figure 2-5. Components of FDAS Utility Subsystem

Computer Operating System described by Booch (1987) would become Flight_Computer_Operating_System_Configuration_Management.) What was needed was a different symbol to easily distinguish packages from subsystems. Such a symbol (an elipse) was introduced by the GOADA team (Figure 2-6).

The use of subsystems as a design entity has eliminated the rationale for nesting Ada packages inside other Ada packages and, as such, has eliminated the need for the use of subprogram call-throughs. This, in turn, allows the design of loosely-coupled, modular systems, enhances localization, permits the development of verbatim reusable components, and minimizes recompilation overhead associated with the inevitable changes that occur during development.

2.3 TYPE PACKAGES

Ada package specifications have proven to be useful mechanisms for modularizing the declaration of types and constants that are used by various library units. Although Types packages were used somewhat in GRODY, they were not specified in the design documentation. This information was deemed necessary by FDAS; therefore, Build 2 Types packages were placed on those particular pages of the design notebook where all or almost all of the entities on the page referenced these packages (Figure 2-7). As a result, it was easy to determine which design components might be affected by a change in a particular Types package. On GOADA, a specific symbol was introduced for the Types package, borrowed from the symbol Booch (1983) uses to indicate the types exported by a package. This technique has been adopted by GOESIM and UARSTELS (Figure 2-8).

2.4 REUSABLE SOFTWARE COMPONENTS IN DESIGN

The GOADA project introduced the use of two vertical parallel lines near the sides of a package or subprogram symbol to

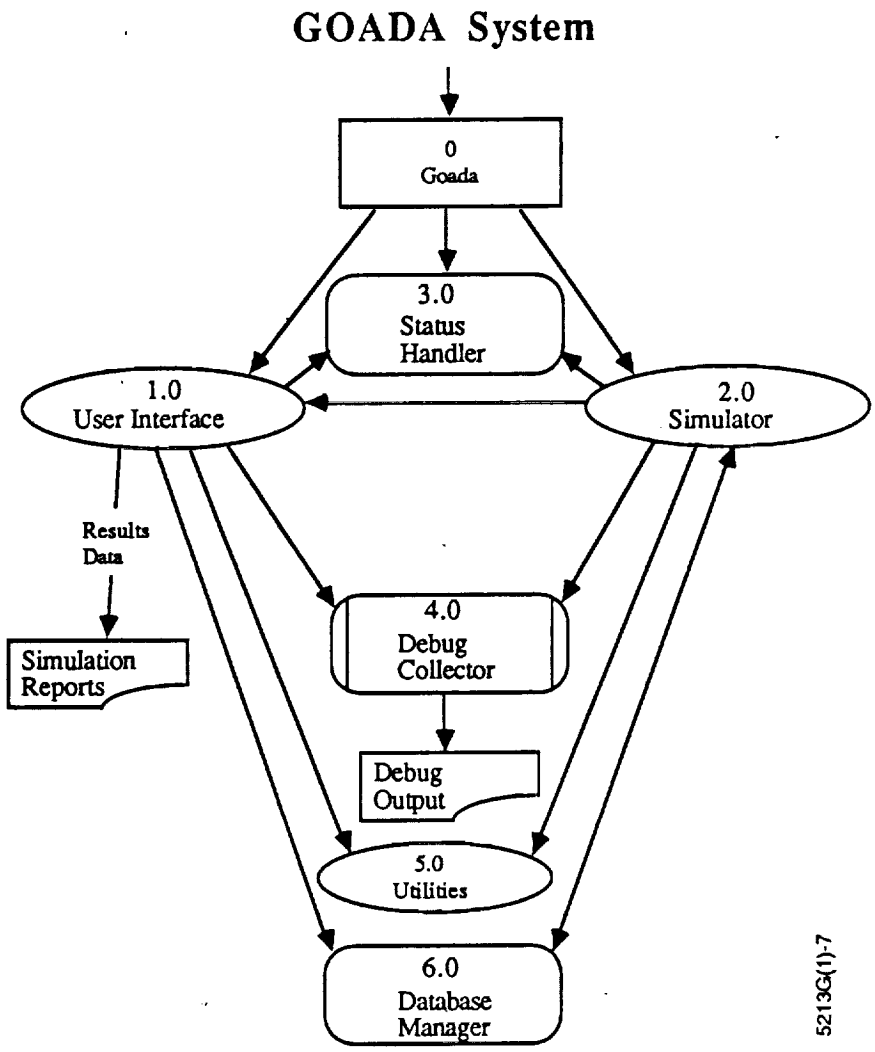

Figure 2-6. Subsystems in GOADA Top-Level Structure Diagram

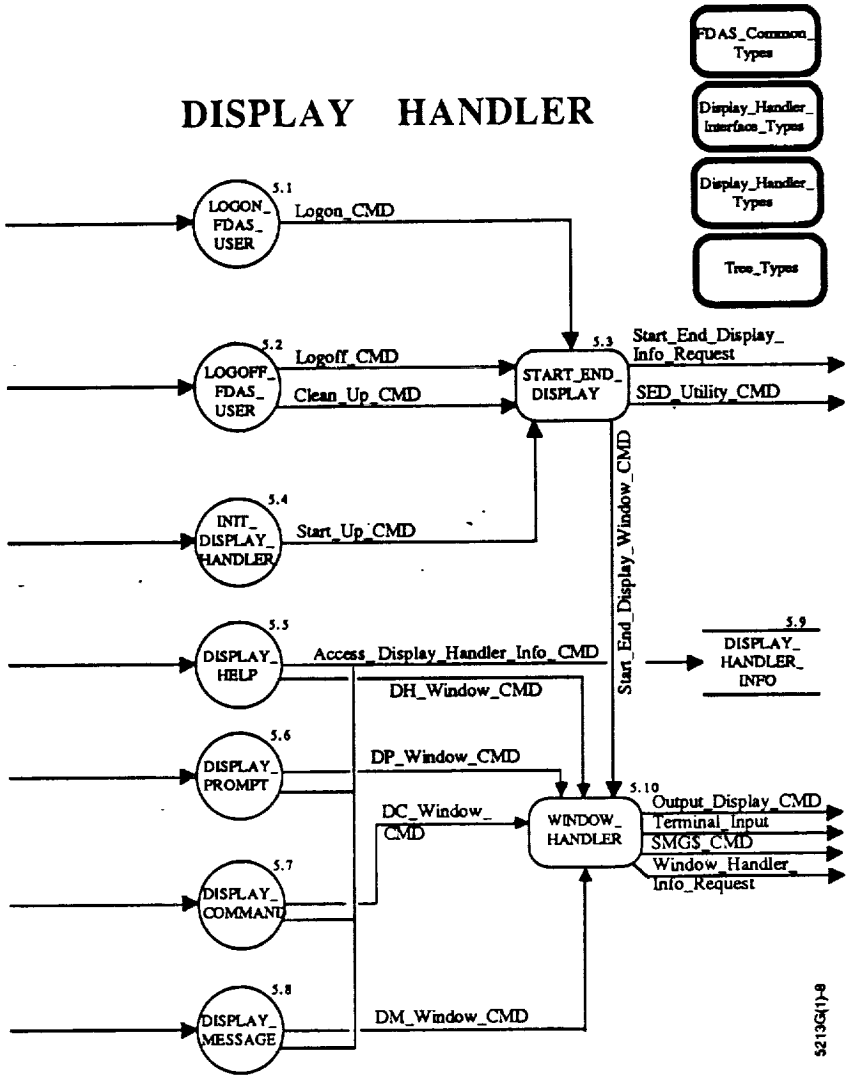

Figure 2-7. Example of Type Package Shown in Design Diagram (Build 2 FDAS)

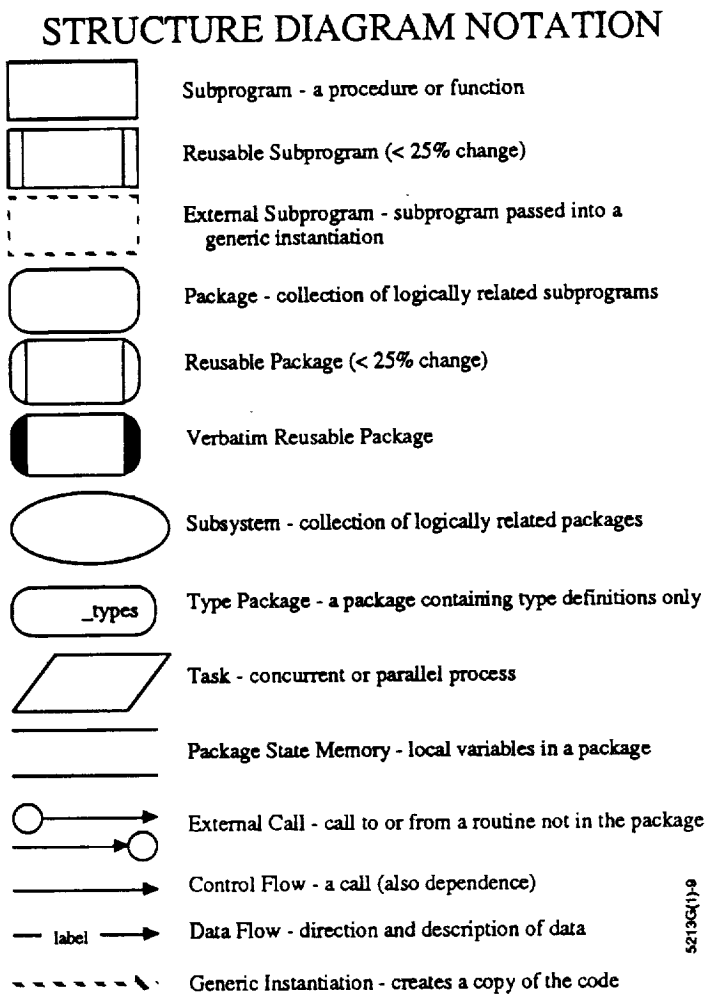

Figure 2-8. Structure Diagram Notation Used on Most Recent Ada Project (UARSTELS)

indicate that the component is being reused. The UARSTELS project expanded on this idea by providing additional information to indicate if a component is being reused with some modifications (less than 25 percent change), or if it is being reused verbatim, i.e., reused with no modification to the source code of the component (Figure 2-8).

An increasing emphasis on the verbatim reuse of Ada components in design has resulted in additional symbols associated with generic packages. The GRODY project introduced the use of a dotted arrow to point from the instantiation of a generic package to the actual generic package that was used in the instantiation. (An example of this from GOADA is shown in Figure 2-9.) This is consistent with the idea that the generic instantiation is dependent on the generic itself. The GOADA project introduced the use of a dotted box to indicate a subprogram that is passed as a parameter into a generic instantiation. (Figure 2-10 shows example from UARSTELS.)

The additional symbols for type packages, reused components, generic packages, and generic instantiations continue the evolutionary process of directly mapping an Ada design entity into a specific type of Ada software component. This evolution should simplify the developers' task of using the design document to implement the design of the system.

2.5 COMPILED DESIGN

The GRODY project investigated the concept of developing compilable design elements during the design phase of the software development life cycle. However, only a small portion of the system was actually compiled by the time of the CDR. The majority of the package specifications were compiled early in the implementation phase, including some

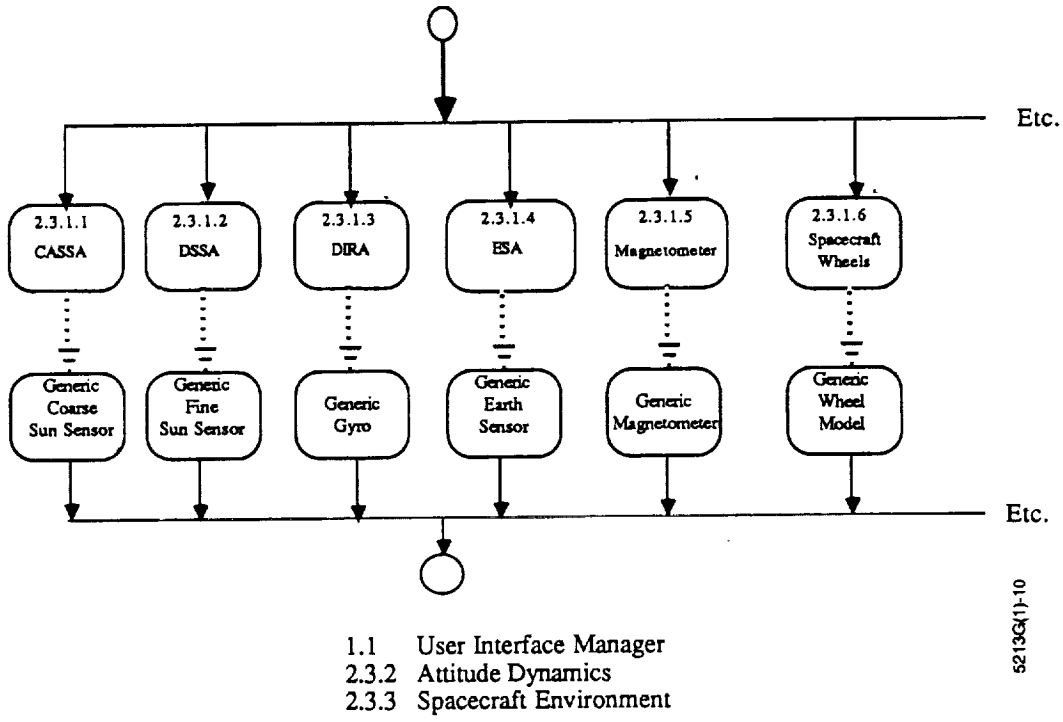

Figure 2-9. Example of Generic Package and Instantiation From GOADA Project

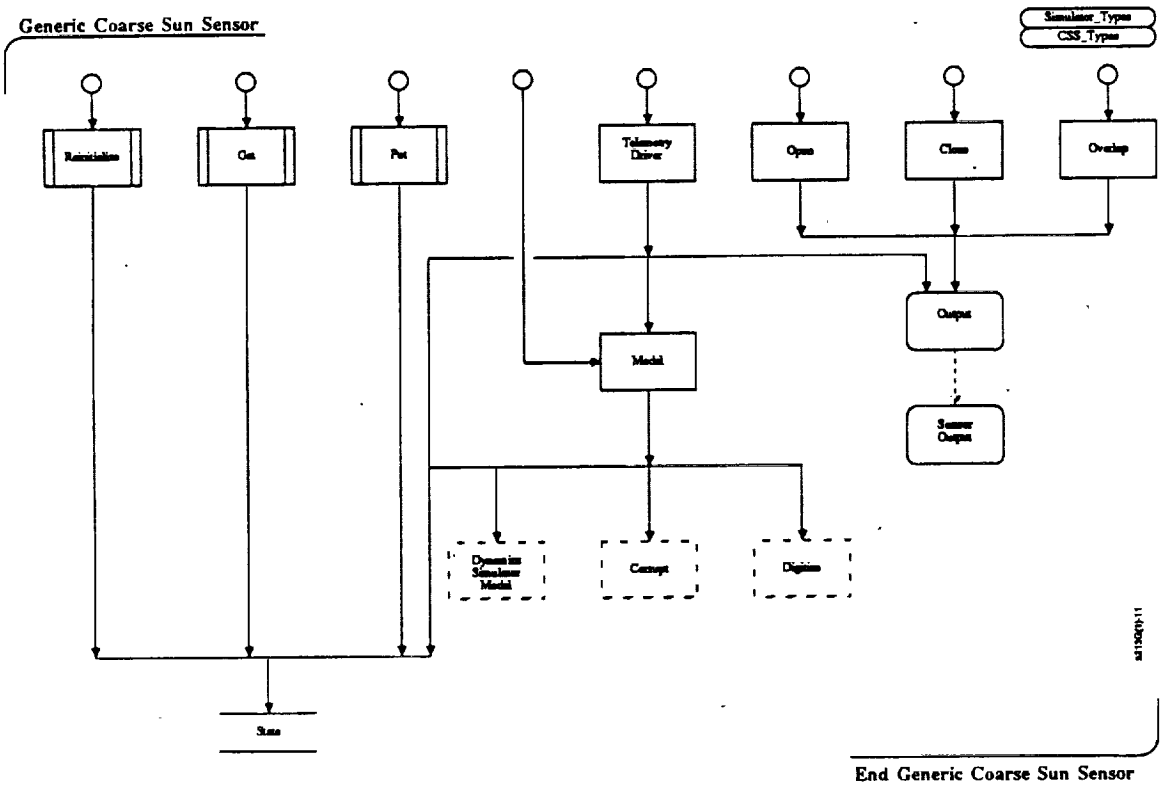

Figure 2-10. Design Diagram From UARSTELS Showing Subprograms as Parameters for Generic Instantiation

Ada PDL. Most of the GRODY team members felt that this compilation of software components and PDL should be considered as a design activity, with the compiler being used essentially as an interface and type-checking tool to verify consistency across the project (Godfrey and Brophy, 1987).

This idea from GRODY of developing compiled design elements during the design phase was adopted by all subsequent Ada projects, and the concept has been expanded to include development of a compiled design. This is a more rigorous concept because the term here has been defined to mean that all of the compiled design elements that make up the system must be sufficiently complete such that the entire system can be successfully linked into an executable image.

A design in Ada can be compiled to different levels of detail:

1. Compilation of the specifications and implementations (bodies) of the packages within the system plus compilation of the program driver. At a minimum, this requires that the subprogram bodies implemented within the package bodies contain a null; statement:

```
package body Generic_Thruster is

  ...

  procedure Model (
   Thruster_ID:  in THRUSTER_ID_TYPE) is

  begin

   null;

  end Model;

  ...

end Generic_Thruster;
```

2. Same as (1) above, except each subprogram in each package body is separated into individual files and compiled, leaving behind in the package bodies only the stubs of these subprograms:

```
package body Generic_Thruster is
  ...
  procedure Model (
    Thruster_ID:  in THRUSTER_ID_TYPE)
    is separate;
  ...
end Generic_Thruster;
separate (Generic_Thruster)
procedure Model (Thruster_ID:  in THRUSTER_ID_TYPE)
is
begin
  null;
end Model;
```

3. Same as (2) above, except commented out PDL statements are included in the body of the subunits

4. Same as (3) above, except control statements are compiled (loop ... end loop;, if ... then ... end if;, case ... end case;)

5. Same as (4) above, except commented out calls to lower-level subroutines are included

6. Same as (5) above, except calls to lower-level subroutines are compiled

In practice, most of the newly designed software components for GOADA, GOESIM, and UARSTELS have been compiled to the

level indicated in (4) or (5) above. This provides interface checking among all of the compilation units that comprise individual packages and syntax checking of the control statements within subunits. However, utilizing the compiler to check interfaces across package boundaries requires compiling to the level indicated in (6) above. In this case, all calls made to subprograms within lower-level packages referenced by a particular unit are coded and compiled. This requires that all variables used as actual parameters (arguments) in subprogram calls must be declared, and the types of these variables must be identical to the types of the formal parameters associated with the called subprograms. Since most of the executable code in non-terminal subprograms often consists of control structures wrapped around subprogram calls, many developers felt that units compiled to this level would be (for the most part) implemented before the implementation phase of the project had actually started. As a result, most developers indicated that a design should be compiled to the level that included compiled control statements within the PDL. The average level to which a design should be compiled, by project is

GRODY	GOADA	GOESIM	UARSTELS
3	4.4	5.2	4

The introduction by the GRODY project of the concept of developing compilable design elements during design has been expanded by the production simulator projects to include the development of a compiled design.

2.6 PROJECT REVIEWS

All four of the simulator projects have followed the traditional approach utilized on FORTRAN systems of having two

formal design reviews, a PDR, and a CDR. Presently no reason appears to suggest that a different approach is required in this environment for an Ada project although consideration should be given to making some modification to the time these reviews are held during the project life cycle, as discussed below.

2.6.1 PRELIMINARY DESIGN REVIEW

Except for the specific references to structured design and FORTRAN language constructs, all four of the simulator projects followed the steps outlined in the Recommended Approach To Software Development (McGarry et al., 1983) during preliminary design. Thus, for each project the high-level architecture of the system was defined, and each top-level subsystem was refined to two additional levels of abstraction. The entity diagram notation was used to represent this design within the preliminary design report, which is the primary product of preliminary design. For the three production simulator projects, the Ada package specifications defined for the entity diagrams were designed, coded, and compiled, a process that is specific to using Ada. Finally, the design was subjected to formal management and technical review through the PDR. All four task leaders of the simulator projects indicated that the PDR was helpful as a part of the design phase of the software development life cycle.

2.6.2 CRITICAL DESIGN REVIEW

The development of a detailed design for each of the production Ada simulators also followed the traditional approach used on the FORTRAN systems, with the major exception that the design was compiled by the time of the CDR. The primary product developed during this phase was the detailed design document, which was produced by continually refining into greater detail the entity diagrams generated for the

preliminary design document. The other major product of detailed design was the code and PDL produced during development of the compiled design. For this part of the design phase, the package specifications compiled before the PDR were used as input to an in-house utility called Package_Helper, which automatically generated compilable package bodies and compilable subprogram subunits as described above. The bulk of the remaining design work then involved developing PDL within each of the subunits, and then compiling these units into the appropriate Ada library. Once these components were compiled, the entire system was linked into an executable image.

The CDR appears to be a suitable, sufficient approach to formal review of the products of the detailed design effort by management and technical personnel. However, a number of developers expressed the opinion that the CDR on their project was held too soon after the PDR. These developers suggested that the schedule pressure to produce large amounts of compilable code and PDL by the time of the CDR did not allow a sufficient amount of time to think through the details of the design.

SECTION 3 - PROJECT CHARACTERISTICS

3.1 PLANS AND ESTIMATES

As a part of the planning process, task personnel for each of the three production simulators estimated the total staff effort hours that would be required for the duration of the project based on prior experience with FORTRAN projects and the manager's guidelines for the Flight Dynamics area. During the requirements analysis and design phases of the project, GOADA required 8,144 hours, or 35.4 percent of the total hours estimated; GOESIM required 4,218 hours, or 32.6 percent of the total hours estimated; and UARSTELS required 3,008 hours, or 29.5 percent of the total hours estimated for the project. In comparison, the manager's guidelines, developed in the Flight Dynamics area for planning FORTRAN projects, suggest that the staff hours needed for requirements analysis and design should be 30 percent of the total effort for a project. The 30 to 35 percent estimated figure for the three Ada projects is similar to the 30 percent figure used for the FORTRAN projects, but this may be because these Ada projects were planned using the same estimation techniques applied to FORTRAN projects and required to adhere to these planned schedules. For the GOADA project, most developers felt that insufficient effort was allowed for the detailed design, and those portions of the system that had not been fully designed by the time of the CDR were completed early in the implementation phase. Therefore, the 35 percent estimated effort for design for GOADA is likely to be understated.

The Flight Dynamics area also traditionally estimaces anticipated life cycle phase start and end dates as a part of project planning. Task personnel on the three production simulator projects each took a different approach in

estimating what amount of calendar time the design phase would require as a percentage of the project life-cycle. On the GOADA project, 36.5 percent of the elapsed time of the project was planned for requirements analysis and design. On GOESIM and UARSTELS, this percentage was 46 percent and 44 percent, respectively. By CDR, the planned dates had changed somewhat. GOADA had replanned, slipping the end date of system test by 1 month. GOESIM went to CDR 2 weeks late but adhered to the remainder of the schedule. UARSTELS went to CDR as originally planned. As it turned out, all three projects eventually allocated similar percentages of elapsed time to the requirements analysis and design phases: GOADA, 43 percent; GOESIM, 49 percent; and UARSTELS, 44 percent.

In comparison, the guidelines developed for FORTRAN allocates 35 percent of the time scheduled for requirements analysis and design. This is 10 to 15 percent lower than what was experienced on the three Ada projects as of CDR. It will be interesting to see how or if the schedule changes after the implementation phases are completed on each of the three projects.

3.2 DEVELOPMENT ACTIVITIES THROUGH DESIGN

The profiles of these projects differ in terms of the distribution of effort among the following categories of development activities up to the time of the CDR:

- Predesign (PREDES)
- Create design (CREDES)
- Read and review design (RDREVDES)
- Coding (CODE)
- Other activities (training, meetings, technical management (OTHER)

- Support (Program management, technical publications, librarian, secretarial) (SUPPORT)

The research and development orientation of GRODY is apparent from this project's activity profile (Figure 3-1). Nearly half (48.3 percent) of the effort was charged to the OTHER category, much of which included Ada training and work on developing a design methodology suitable for a system of this size.

The next three projects that followed in time were the GOADA, GOESIM, and UARSTELS production projects. Since most of the technical groundwork on design issues related to Ada had been worked out by GRODY, these projects were able to devote the largest percentage of their activity to actually creating the design, and this percentage was very similar across all three projects--41.1 percent for GOADA, 42.9 percent for GOESIM, and 38.3 percent for UARSTELS (Figure 3-1).

3.3 REUSE

The GRODY project had no flight dynamics software written in Ada to draw from and, as such, were not able to reuse any Ada code; the 3.7 percent reuse reported by GRODY was limited to imported FORTRAN procedures obtained from a previous dynamics simulator (Figure 3-2). For the production Ada simulator projects that followed, a concerted effort was made to reuse this Ada software from GRODY. The component reuse percentages for the later projects were 42 percent for GOADA, 30 percent for GOESIM, and 50 percent for UARSTELS (Figure 3-2).

Unfortunately, the GRODY team did not well understand design considerations necessary for the implementation of Ada software components that could be readily reused on future Ada projects. This problem can be viewed in part as a result of

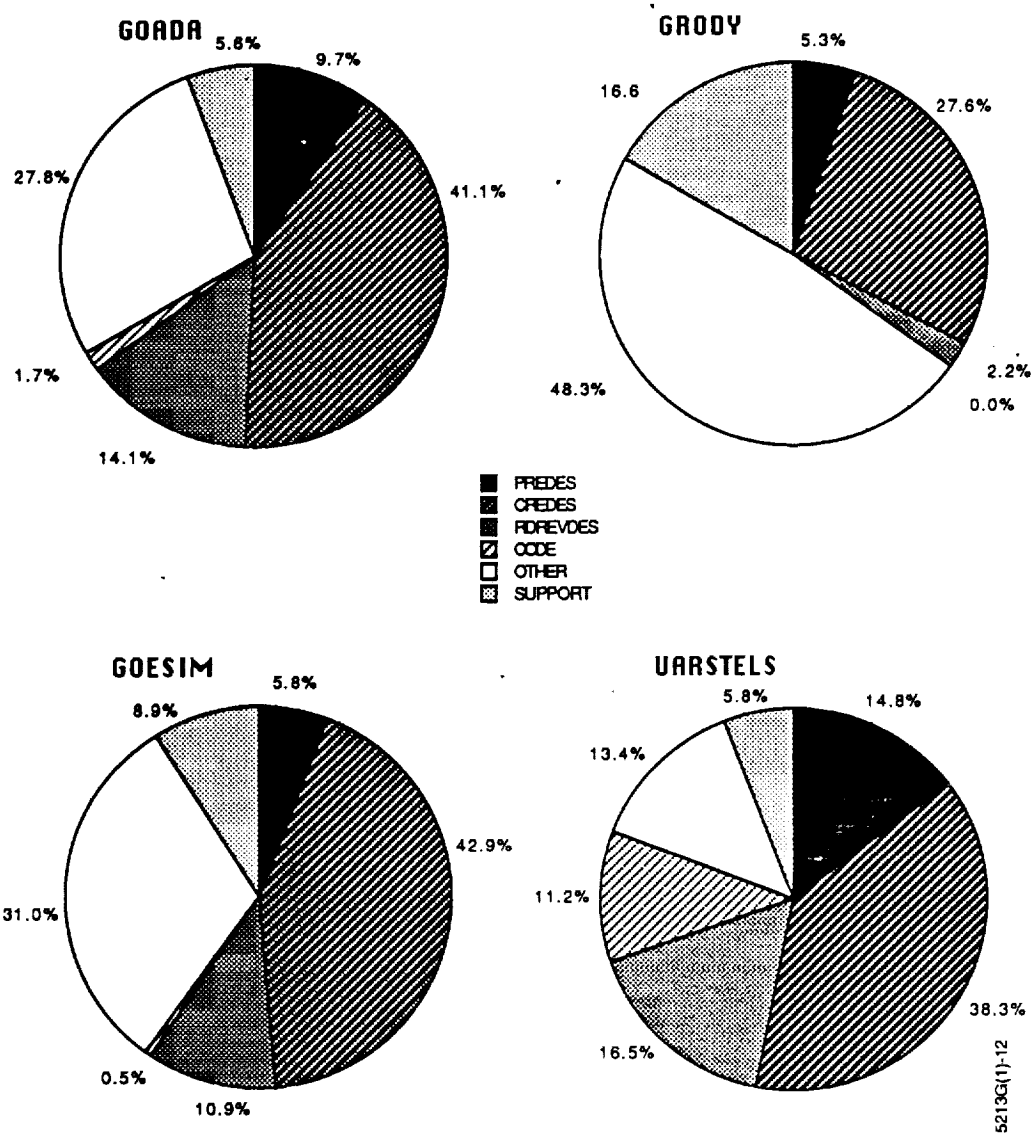

Figure 3-1. Distribution of Effort Over Activities During Design Phase

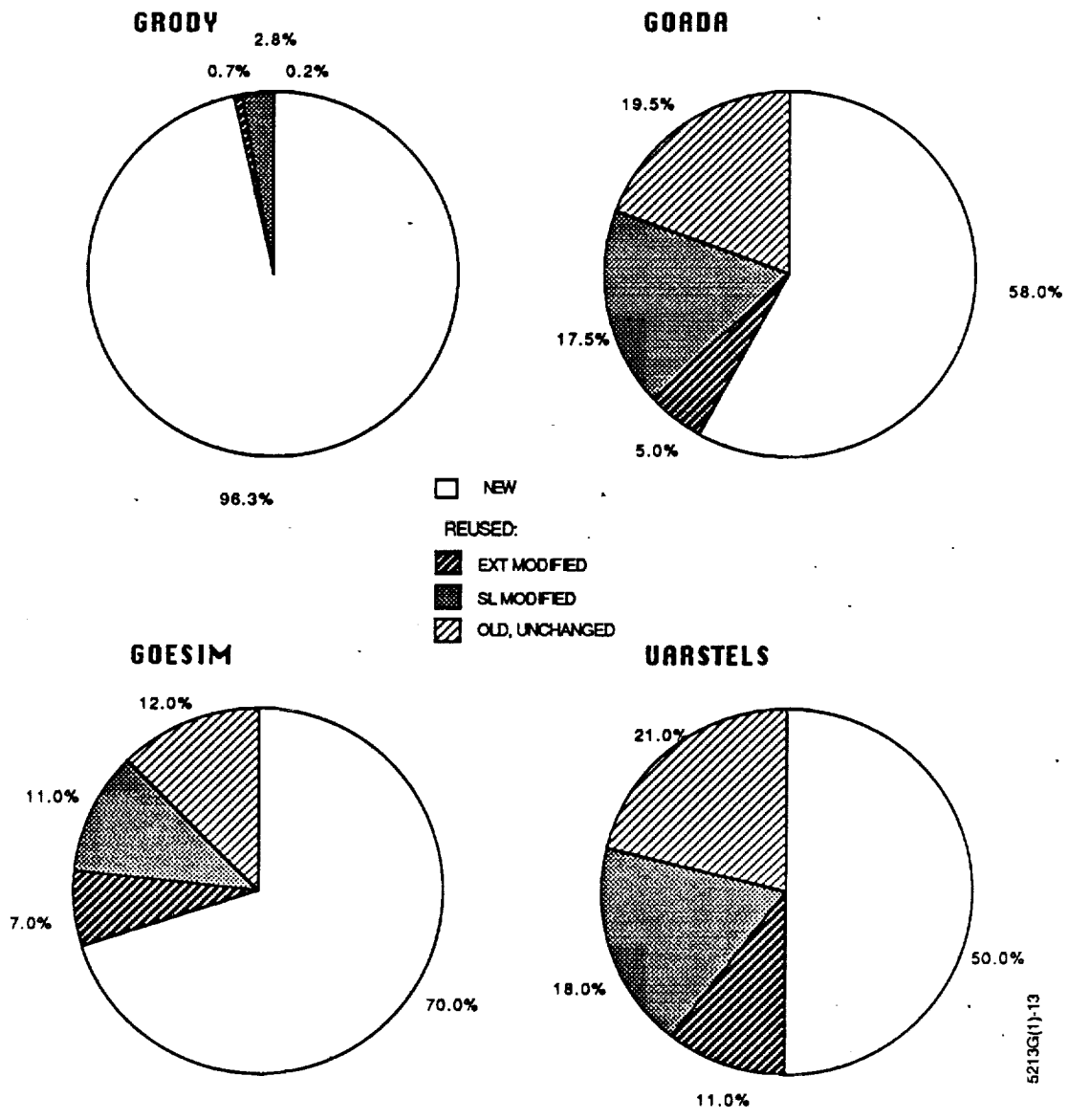

Figure 3-2. Component Reuse on Ada Simulator Projects

the team's lack of the use of subsystems as a design entity and an inappropriate use of nesting to enforce component visibility (Clarke et al., 1980). As a result, during the development of GOADA's compiled design, a considerable amount of effort was needed to extract software components from the heavily nested components of GRODY and then to reconstruct these components into smaller, individual library units. This extra effort can be seen in Table 3-1, which shows the percentage of the total effort during the design phase spent on four special activities: documentation, enhancement and optimization, reuse of software, and rework needed as the project progressed through design. During the design phase, 8.1 percent of the total effort during the design phase was spent on reuse of software for the GOADA project as compared to 4 percent for the GOESIM project. The effort expended on reuse increased somewhat on UARSTELS (5.7 percent), primarily because many of the packages available from GOADA were modified into generic packages.

Table 3-1. Percentage of Total Effort Spent on Special Activities During Design

	GRODY	GOADA	GOESIM	UARSTELS
Documentation	ND*	19.7	11.8	13.3
Enhancement and Optimization	ND	4.4	5.8	.9
Reuse of Software	ND	8.1	4.3	5.7
Rework	ND	1.7	3.5	.8

*ND = no data available

The GOESIM project planned to reuse many of the components un-nested by the GOADA developers. However, as apparent by the effort needed to rework software, some project-specific

dependencies were found within many of these reused components. For example, the Spacecraft_Ephemeris component from the GOADA project logs error data by calling User_Interface.Receive_Error. Since telemetry simulators are batch programs that typically do not have a user interface, all of these calls had to be modified to calls to the Error_Collector.

3.4 UTILIZATION OF CPU RESOURCES IN DESIGN

Traditional FORTRAN projects utilize text editors on a computer system to enter subprogram PDL, COMMON blocks, and NAMELISTs before PDR and CDR. The FORTRAN compiler is not utilized during design on these projects. Compared to a traditional FORTRAN project, central processing unit (CPU) resource consumption can be expected to be much higher during the design phase of an Ada project that generates a compiled design. Two major reasons for this increase are

- The Ada compiler is being used extensively during the design phase.
- Ada compilers are typically large, complicated programs that require substantial CPU resources.

Figure 3-3 shows the profile of CPU usage over the design phases of the GOESIM and UARSTELS projects. The first, smaller peak on the UARSTELS project occurred near the PDR. The second peak on UARSTELS occurred several weeks before the CDR because the project was somewhat ahead of schedule. The large peak on GOESIM occurred near the CDR.

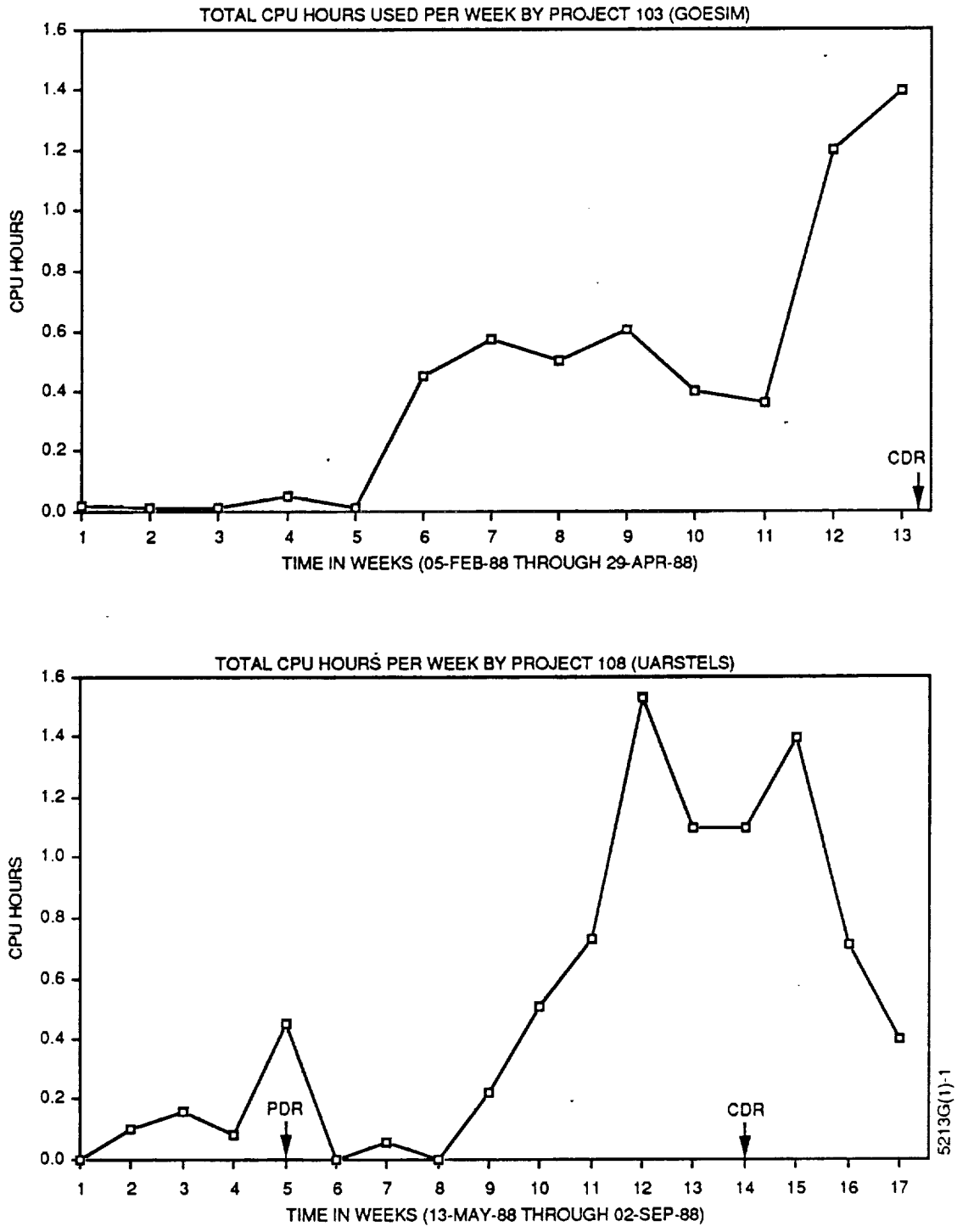

Figure 3-3. Profile of CPU Utilization During Design on GOESIM and UARSTELS

3.5 SOFTWARE METRICS

The development of a linkable, compiled system during the design phase in Ada is a process similar to implementation since a considerable amount of attention must be given to details that previously have not been addressed by traditional FORTRAN design teams. As a result, it makes sense to begin tracking the progress of Ada software projects early in the design phase, using software metric tools that traditionally have been utilized only during implementation and testing.

One major advantage of developing a compiled design in an Ada project is that it represents a clearly defined milestone. If a consistent definition of the term is adopted across all Ada projects, existing effort data and software metrics such as source lines of code (SLOC), delivered source instructions (DSI) or component counts can be used to compute productivity during design. These productivity measures can then be used to determine, for example, if reported levels of software component reuse in design are correlated with actual increases in productivity during design, if higher productivity in design is correlated with a reduction in overall system cost, etc.

The GOADA, GOESIM, and UARSTELS projects, all produced compilable designs and were compiled to approximately the same level of detail. (The GRODY project did not develop a compiled design.) The compiled designs of the production simulators are referred to as Build 0. Although the GRODY team developed a Build 0 (well after CDR), major parts of the system had not been designed yet, and the system could not be linked.

Table 3-2. Software Measures for Build 0

	GRODY	GOADA[1]	GOESIM	UARSTELS
SLOC	19,513	90,507	52,831	49,836
LOC&C	ND	67,684	39,134	39,209
LOC (DSI)	ND	36,979	19,338	21,350
Comments	ND	30,705	19,796	17,859
Blank lines	ND	22,823	13,697	10,627
Statements	ND	12,500	7,212	8,224
Declarative	ND	7,000	4,282	4,636
Executable	ND	5,500	2,930	3,588
Number of components	118	592	410	443

[1]The data on GOADA was collected about 10 days after the CDR; about 2,500 SLOC had already been removed from the Build 0 library into the Digital Equipment Corporation (DEC) Configuration Management System (CMS) library. These 2,500 SLOC are not included in the numbers given for GOADA.

Definition of Terms

SLOC	Source lines of code => a count of carriage returns (<CR>) in the file
LOC&C	Lines of code plus comments => lines containing actual code and comment lines
LOC	Lines of code => lines containing actual code
DSI	Delivered source instructions => same as LOC
comments	Lines that begin with comment token, "--"
blank lines	Lines that contain only a <CR>

Considerable care must be taken in evaluating lines of code as a software productivity measure, particularly for a compiled design. For example, the DSI may not be an appropriate measure for the design phase since it excludes all blanks and comment lines. Because a large fraction of the PDL for these

systems consists of comments, DSI can be expected to somewhat underestimate the size of a compiled design. Support for this idea can be seen when blanks, comments, and executable statements, as a percentage of SLOC for Build 0 of GOADA and GOESIM, are compared with the same percentages for the completed GRODY project.

Table 3-3. Line Count Profiles

	Final Build	Build 0 at CDR		
	GRODY	GOADA	GOESIM	UARSTELS
Blank lines (percent)	26.0	25.2	25.9	21.3
Comments (percent)	27.8	33.9	37.5	35.8
LOC (DSI) (percent)	46.2	40.9	36.6	42.9
	100.0	100.0	100.0	100.0
LOC&C (percent)	74.0	74.8	74.1	78.7

Note that the percentage of blank lines is virtually the same across GRODY, GOADA, and GOESIM, about 25 to 26 percent. However, for the nonblank lines, the ratio of DSI to comments for the combined GOADA and GOESIM projects at Build 0 is 1.12, whereas this ratio is 1.66 for the final build of GRODY, nearly 50 percent higher. This difference can be expected because during implementation, some commented PDL statements are modified into executable code, and additional executable statements are being added to provide the functionality summarized in comments written during design.

Interestingly, approximately 60 to 75 percent of the estimated final SLOC had been completed by the time of the

CDR (Table 3-4). However, as mentioned above, this code contains a considerable fraction of commented PDL statements whose syntax, semantics, and logic are not subject to the rigorous examination of the Ada compiler. In other words, the effort involved in developing the remaining 25 to 40 percent of the system during implementation and testing can be expected to be greater per line of code than the effort involved during design. This hypothesis will be examined in the second report in this series.

Table 3-4. Estimated SLOCs Completed as of CDR

	GOADA	GOESIM	UARSTELS
SLOC at CDR	90,507	52,800	49,836
Estimated SLOC for final system as of October 24, 1988	145,000	78,000	65,000
Percentage recent estimated SLOC completed	62.4%	67.7%	76.7%

3.6 PRODUCTIVITY

The total effort for developing Build 0 for the four simulator projects is shown below:

Table 3-5. Total Effort for Build 0

	GRODY	GOADA	GOESIM	UARSTELS
Staff-hours	6,430.0	8,144.0	4,218.0	3,008.0
Staff-days[1]	803.8	1,018.0	527.3	376.0

[1] A staff-day is 8 staff-hours.

The average productivity measures for each of these projects is indicated below.

Table 3-6. Productivity Measures During Design

	GRODY	GOADA	GOESIM	UARSTELS
SLOC/staff-day	24.3	88.9	100.1	132.5
LOC&C/staff-day	ND	67.8	74.2	104.3
DSI/staff-day	ND	36.3	36.7	56.8
Statements/staff-day	ND	12.3	13.7	21.8
Declarative	ND	6.9	8.1	12.3
Executable	ND	5.4	5.6	9.5
Components/staff-day	0.15	0.58	0.78	1.18

3.7 ADA EXPERIENCE OF DESIGN TEAM

When the number of years of professional experience in developing software in any language are considered for the Ada developers, these four simulation projects appear very similar, as shown in Table 3-4. Another similarity is that the assistant technical representatives (ATRs) and the technical managers associated with each project all have had experience in developing software in the flight dynamics area. On the other hand, for the design phase of these projects, the percentage of developers who have had previous experience in the application area varies widely, as shown in Table 3-4. Similarly, this table also shows a wide variation as to the percentage of design personnel who have had previous professional Ada software development experience.

Table 3-7. Experience of Ada Developers

	GRODY	GOADA	GOESIM	UARSTELS
Software Development Experience (years)	4.7	5.9	5.7	5.5
Ratio of Personnel Experienced in Application	1/7	2/7	1/4	3/3
Ratio of Personnel Experienced in Ada	0/7	3/7	1/4	1/3

As shown in Table 3-8 below, only the GOADA project had a technical manager who has had actual Ada software experience. Both GOADA and UARSTELS were staffed with a task leader who has had previous satellite simulation development experience. Only UARSTELS was staffed with a task leader with previous Ada experience previous satellite simulation development experience, and Ada experience (including satellite simulation development experience in Ada).

Table 3-8. Ada Experience of Project Management

	GRODY	GOADA	GOESIM	UARSTELS
Ada Experience of Technical Manager	no	yes	no	no
Application Area Experienced of Task Leader	no	yes	no	yes
Ada Experience of Task Leader	no	no	no	yes

This general lack of previous Ada experience during the design phase of these projects is due to the relatively recent introduction of Ada and the even more recent introduction of a production-quality Ada compiler for the VAX. The few developers who have had previous Ada experience were drawn from the first two GSFC/CSC Ada projects. The three developers on GOADA with Ada experience came from FDAS; the one developer on GOESIM with Ada experience came from GRODY; and the one developer (also the task leader) on UARSTELS with Ada experience worked on both FDAS and GOADA. The technical manager of GOADA received his Ada experience from development work on GRODY.

Since each Ada project gains from both the mistakes and the technical advances made by previous Ada projects, it is difficult to separate the effect of individual team members' application area experience, training, and professional Ada development experience from the effect produced by this accumulating project legacy. Even so, the one project characteristic that appears to have a major effect on productivity is the presence of a technically strong task leader, with professional Ada development experience in the application area.

3.8 TRAINING

The GRODY team had the widest range of different types and forms of training in Ada. Of these, the lectures in PAMELA (Cherry, 1985), the classroom lectures on Ada syntax and semantics, and the Alsys Ada video training course were rated as having only moderate usefulness. In contrast to these, the practice project was rated as extremely useful by almost everyone on the team, and having actual project experience

as a form of training was also highly rated (OJT = on-the-job-training):

Average rating of the usefulness of each type of training provided on GRODY, on a scale of 1 (not useful) to 9 (extremely useful):

PAMELA	Lecture	Tapes	Books	OJT	Practice Project
4.4	5.3	5.4	7.0	8.3	8.9

The GOADA team was provided with classroom lectures on the syntax and semantics of Ada, object-oriented design, and the use of software development tools. The average rating of the training they received was only slightly higher than that given by the GRODY team for their lectures:

Average rating of the usefulness of the training provided on GOADA, on a scale of 1 (not useful) to 9 (extremely useful):

Lecture	Books
6.1	6.1

One particular criticism of the GOADA training was that the timing of the lectures was not well coordinated with the project schedule. For example, several developers suggested that the lectures on the DEC software development tools should have been late in the lecture series rather than early.

Even with the less than enthusiastic rating of classroom lectures given by these two teams, most developers across all the simulator projects recommended that 40 or more hours of classroom lectures should be provided on the Ada language:

Average number of recommended hours of classroom/lecture Ada training by project:

GRODY	GOADA	GOESIM	UARSTELS
53	43	28	53

SECTION 4 - SUMMARY AND RECOMMENDATIONS

Before these initial Ada projects, almost all software at GSFC/CSC has been developed using FORTRAN, a process that is well understood in the Flight Dynamics environment. Conversely, the development of software systems in Ada for the types of applications found in this environment is a not yet as fully an understood process. An understanding of how to apply Ada technology more effectively in the Flight Dynamics environment can be expected through the accumulated experience gained from these projects and from future Ada projects.

The transition from developing software systems in FORTRAN to developing systems in Ada is an evolutionary process. When the first Ada projects were started, no really well defined design methodology existed for use on production-level software systems that allowed the design of systems that effectively utilized the data abstraction capabilities of the Ada language. As a result, an object-oriented design methodology and a graphical design notation were developed in-house for use on Ada systems; this design technique has been incrementally enhanced and refined by subsequent Ada projects. The lack of a subsystem concept for the GRODY project resulted in the development of a tightly coupled system of large compilation units that were not directly reusable by follow-on Ada projects. The production simulator projects all utilized the subsystem concept in their designs and extended the GRODY project's recommendation to produce compilable package specifications in developing a compiled design. The project UARSTELS greatly increased the use of Ada generic packages in design and was the first project to emphasize during design the development of software components that could be reused on subsequent projects without changes.

The design team for an Ada project should have a mixture of personnel with different areas of expertise and experience. These areas include flight dynamics, Ada software system development, mathematics, and specific application-area experience. In particular, the presence of a technically strong task leader with professional Ada development experience in the application area may be the single most important factor in producing a well designed system within schedule and budget.

The incremental design approach used on the production Ada projects meshes well with the existing PDR/CDR approach that has been utilized on traditional FORTRAN systems. Some consideration should be given to extending the length of the detailed design phase to compensate for the additional effort needed to generate the large amount of code associated with developing a compiled design.

All developers expressed a desire for formal training in Ada syntax and semantics, Ada software development tools, and Ada design methodologies. Many of these developers suggested that this training should be spread out over time, instead of a concentrated training presented over a few days. In addition, this training should be carefully coordinated with the project schedule to maximize its effectiveness, particularly for training in software development tools.

Additional thought needs to be given to how to more effectively exploit features of Ada in a manner that will maximize the amount of reusability of Ada software design components from one simulation project to another. Consideration should be given as to how to develop these design entities so that they are reusable on larger systems, such as AGSSs, and even on very large-scale systems, including the Space Station project.

GLOSSARY

AGSS	Attitude Ground Support System
ATR	assistant technical representative
CDR	critical design review
CMS	Configuration Management System
CPU	central processing unit
CSC	Computer Sciences Corporation
DEC	Digital Equipment Corporation
DSI	delivered source identification
FDAS	Flight Dynamics Analysis System
GOAPA	GOES-I Dynamics Simulator
GOES-I	Geostationary Operational Environmental Satellite-I
GOESIM	GOES-I Telemetry Simulator
GRO	Gamma Ray Observatory
GRODY	GRO Simulator in Ada
GROSS	GRO Simulator Systems
GSFC	Goddard Space Flight Center
NASA	National Aeronautics and Space Administration
PAMELA	Process Abstraction Methodology
PDL	program design language
PDR	preliminary design review
SLOC	source lines of code
UARS	Upper Atmosphere Research Satellite
UARSTELS	UARS Telemetry Simulator

REFERENCES

Agresti, W. W., et al., "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

Booch, G., Software Engineering With Ada. Menlo Park, CA: Benjamin/Cummings Publishing Company, 1983

--, "The Architecture of Complex Systems," Software Components With Ada, Chapter 17. Menlo Park, CA: Benjamin/Cummings Publishing Company, 1987

Brophy, C., et al., "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the First International Ada Symposium, Washington, DC, March 1987

Cherry, G. W., "Advanced Software Engineering With Ada--Process Abstraction Method for Embedded Large Applications," Language Automation Associates, Reston, VA, 1985

Clarke, L. A., et al., "Nesting in Ada Programs Is for the Birds", ACM Sigplan Notices, November 1980, Vol. 15, No. 11

Godfrey, S., and C. Brophy, SEL-87-004, Assessing the Ada Design Process and Its Implications: A Case Study, Software Engineering Laboratory, July 1987

Godfrey, S., and C. Brophy, Implementation of a Production Ada Project--the GRODY Study, 1989, in preparation

McGarry, F., et al., SEL-81-205, Recommended Approach to Software Development, Software Engineering Laboratory, April 1983

Myers, G. J., Composite/Structured Design. New York, NY: Van Nostrand Reinhold Company, 1978

Seidewitz, E., and M. Stark, SEL-86-002, General Object-Oriented Software Development, Software Engineering Laboratory, August 1986

Seigle, J., and Y-L Shi, SEL-88-001, System Testing of a Production Ada Project: The GRODY Study, Software Engineering Laboratory, November 1988

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-004, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-78-302, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3), W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-106, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, May 1985

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-203, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo, June 1984

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, August 1983

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1), W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-606, Annotated Bibliography of Software Engineering Laboratory Literature, S. Steinberg, November 1988

SEL-83-001, An Approach to Software Cost Estimation, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-002, Configuration Management and Control: Policies and Procedures, Q. L. Jordan and E. Edwards, December 1984

SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, Proceedings From the Ninth Annual Software Engineering Workshop, November 1984

SEL-85-001, A Comparison of Software Verification Techniques, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, R. Murphy and M. Stark, October 1985

SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985

SEL-85-004, Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, R. W. Selby, Jr., May 1985

SEL-85-005, Software Verification and Testing, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, Proceedings From the Tenth Annual Software Engineering Workshop, December 1985

SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

SEL-86-002, General Object-Oriented Software Development, E. Seidewitz and M. Stark, August 1986

SEL-86-003, Flight Dynamics System Software Development Environment Tutorial, J. Buell and P. Myers, July 1986

SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986

SEL-86-005, Measuring Software Design, D. N. Card, October 1986

SEL-86-006, Proceedings From the Eleventh Annual Software Engineering Workshop, December 1986

SEL-87-001, Product Assurance Policies and Procedures for Flight Dynamics Software Development, S. Perry et al., March 1987

SEL-87-002, Ada Style Guide (Version 1.1), E. Seidewitz et al., May 1987

SEL-87-003, Guidelines for Applying the Composite Specification Model (CSM), W. W. Agresti, June 1987

SEL-87-004, Assessing the Ada Design Process and Its Implications: A Case Study, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-005, Flight Dynamics Analysis System (FDAS) Build 3 User's Guide, S. Chang et al., October 1987

SEL-87-006, Flight Dynamics Analysis System (FDAS) Build 3 System Description, S. Chang, October 1987

SEL-87-007, Application Software Under the Flight Dynamics Analysis System (FDAS) Build 3, S. Chang et al., October 1987

SEL-87-008, Data Collection Procedures for the Rehosted SEL Database, G. Heller, October 1987

SEL-87-009, Collected Software Engineering Papers: Volume V, S. DeLong, November 1987

SEL-87-010, Proceedings From the Twelfth Annual Software Engineering Workshop, December 1987

SEL-88-001, System Testing of a Production Ada Project: The GRODY Study, J. Seigle and Y. Shi, November 1988

SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988

SEL-88-003, Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis, K. Quimby and L. Esker, December 1988

SEL-RELATED LITERATURE

Agresti, W. W., Definition of Specification Measures for the Software Engineering Laboratory, Computer Sciences Corporation, CSC/TM-84/6085, June 1984

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," Proceedings of the First Pan-Pacific Computer Conference, September 1985

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," Journal of Systems and Software, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," Proceedings of the International Computer Software and Applications Conference, October 1985

[4]Basili, V. R., and D. Patnaik, A Study on Fault Prediction and Reliability Assessment in the SEL Environment, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," Communications of the ACM, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P--A Prototype Expert System for Software Engineering Management," Proceedings of the IEEE/MITRE Expert Systems in Government Symposium, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press, 1979

[5]Basili, V. and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," Proceedings of the 9th International Conference on Software Engineering, March 1987

[5]Basili, V. and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," Proceedings of the Joint Ada Conference, March 1987

[5]Basili, V. and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, June 1988

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., Comparing the Effectiveness of Software Testing Strategies, University of Maryland, Technical Report TR-1501, May 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, July 1986

[5]Basili, V. and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering (in press)

[2]Basili, V. R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: IEEE Computer Society Press, 1978

[5]Brophy, C., W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the Joint Ada Conference, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," Proceedings of the Washington Ada Technical Conference, March 1988

[3]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, October 1985

[5]Card, D. and W. Agresti, "Resolving the Software Science Anomaly," The Journal of Systems and Software, 1987

[6]Card, D. N., and W. Agresti, "Measuring Software Design Complexity," The Journal of Systems and Software, June 1988

[4]Card, D., N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986

[5]Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," IEEE Transactions on Software Engineering, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," ACM Software Engineering Notes, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: IEEE Computer Society Press, 1983

[5]Doubleday, D., "ASAP: An Ada Static Source Code Analyzer Program," University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S. and C. Brophy, "Experiences in the Implementation of a Large Ada Project," Proceedings of the 1988 Washington Ada Symposium, June 1988

Hamilton, M., and S. Zeldin, A Demonstration of AXES for NAVPAK, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

Jeffery, D. R., and V. Basili, "Characterizing Resource Data: A Model for Logical Association of Software Data," University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," Proceedings of the Tenth International Conference on Software Engineering, April 1988

[5]Mark, L. and H. D. Rombach, "A Meta Information Base for Software Engineering," University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L. and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[5]McGarry, F. and W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," Proceedings of the Hawaiian International Conference on System Sciences, January 1985

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," Proceedings of the Eighth International Computer Software and Applications Conference, November 1984

[5]Ramsey, C. and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," IEEE Transactions on Software Engineering, March 1987

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," Proceedings From the Conference on Software Maintenance, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, January 1989

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," Proceedings of the 21st Hawaii International Conference on System Sciences, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," Proceedings of the CASE Technology Conference, April 1988

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications, October 1987

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," Proceedings of the Joint Ada Conference, March 1987

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

[5]Valett, J. and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," IEEE Transactions on Software Engineering, February 1985

[5]WU, L., V. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," Proceedings of the Joint Ada Conference, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," Journal of Systems and Software, 1988

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

NOTES:

[1]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985.

[4]This article also appears in SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986.

[5]This article also appears in SEL-87-009, Collected Software Engineering Papers: Volume V, November 1987.

[6]This article also appears in SEL-88-002, Collected Software Engineering Papers: Volume VI, November 1988.